Optimizing Performance in iPhone Applications

Session 415

These are confidential sessions—please refrain from streaming, blogging, or taking pictures.

Optimizing Performance in iPhone Applications

David Myszewski
iPhone Performance

What You'll Learn

- Drawing and scrolling
- Application launch
- Memory usage
- Files and data
- Power and battery life

iPhone Hardware

It’s not a Mac

	Slowest Mac Mini	iPhone, iPod Touch	iPhone 3G S
RAM	1GB	128MB	256MB RAM
Mass Storage	120GB, Hard Drive	8–32GB, Flash	16–32GB, Flash
Networking	Ethernet, Wi-Fi	Wi-Fi, 3G, EDGE	Wi-Fi, 3G, EDGE

What You'll Learn

- Drawing and scrolling
- Application launch
- Memory usage
- Files and data
- Power and battery life

Demo

iPhone performance

The Drawing Environment

Drawing technologies

Drawing Optimizations

The preferred way to draw

- Use built-in drawing whenever possible
  - UIKit views are already optimized, e.g., UIImageView, UILabel
  - Take advantage of animatable view properties

Drawing Optimizations

Custom drawing

- Mark views as opaque
- Draw minimally
  - Call `setNeedsDisplayInRect:`
  - Check the rectangle passed to `drawRect:`

Drawing Optimizations

Custom drawing

- Cache static drawing objects
  - But be prepared to release them in low-memory conditions

Drawing Optimizations

Timers

- Common drawing bug
  - Invalidate timers before creating new timers

Drawing Optimizations

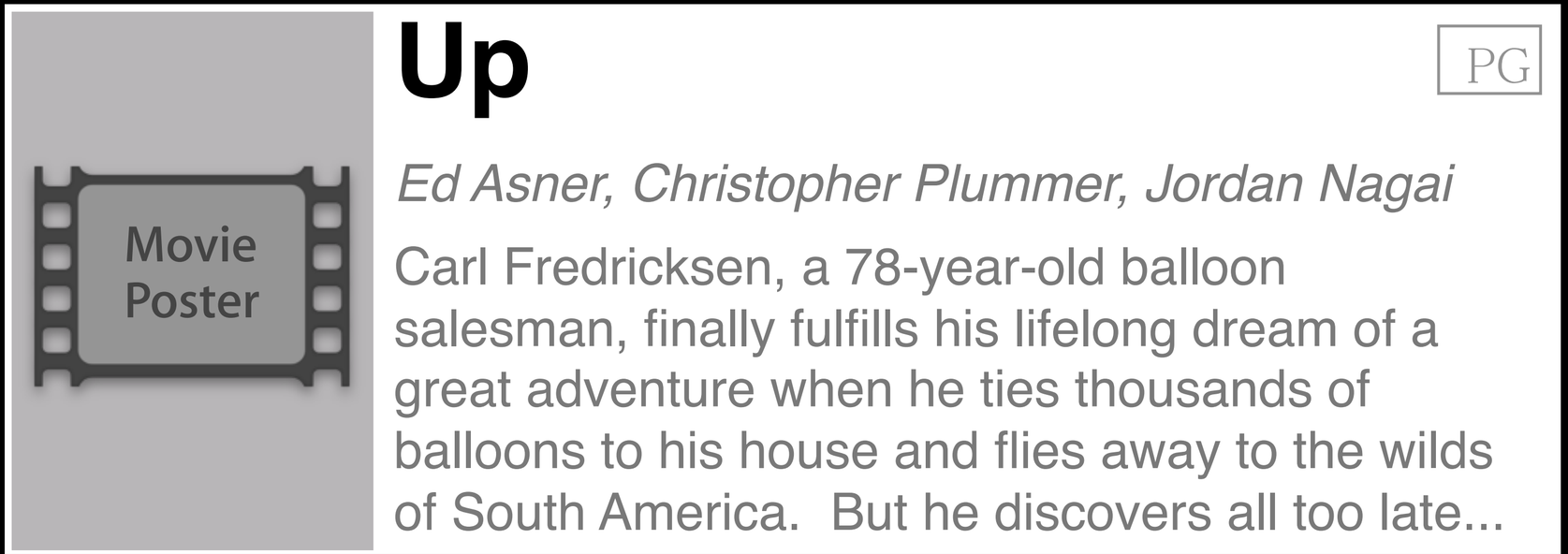
Use PNG files

- Optimized at build time
- Converted from unmultiplied RGBA to premultiplied BGRA
- Unnecessary data removed
- Compression rate may be increased

Drawing Optimizations

Improving your scrolling performance

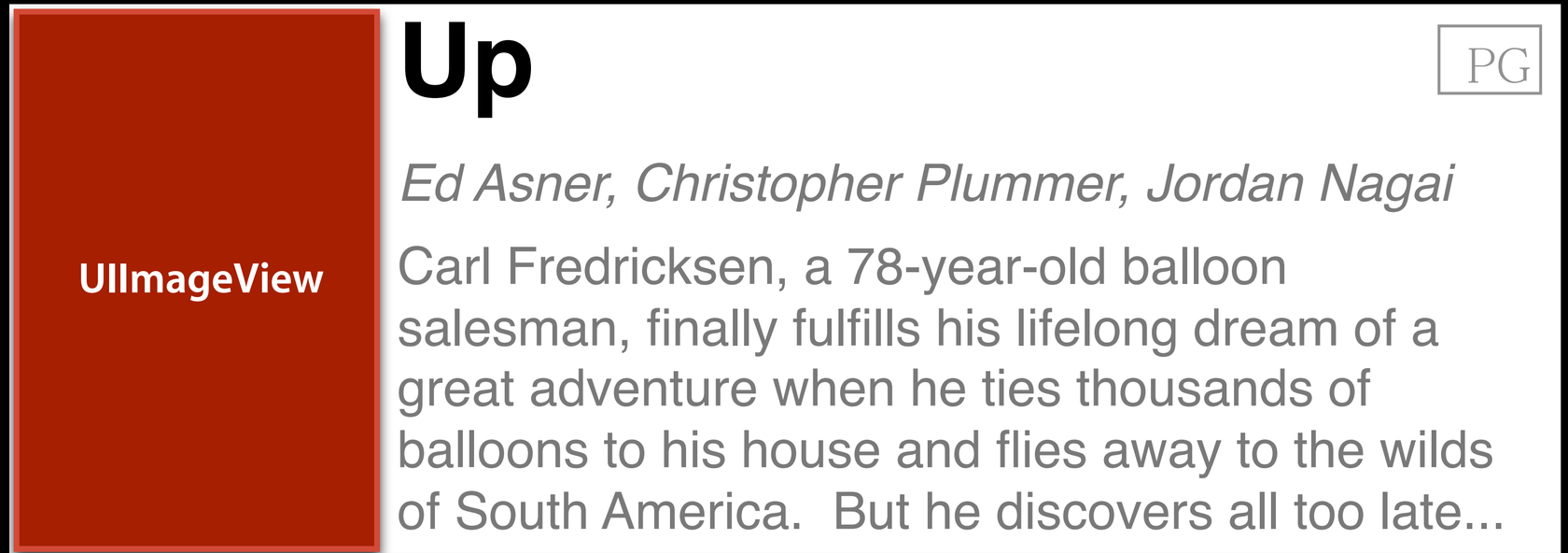
- Use opaque views when possible
- Avoid allocating views during scrolling
  - Allocating views is expensive



Drawing Optimizations

Improving your scrolling performance

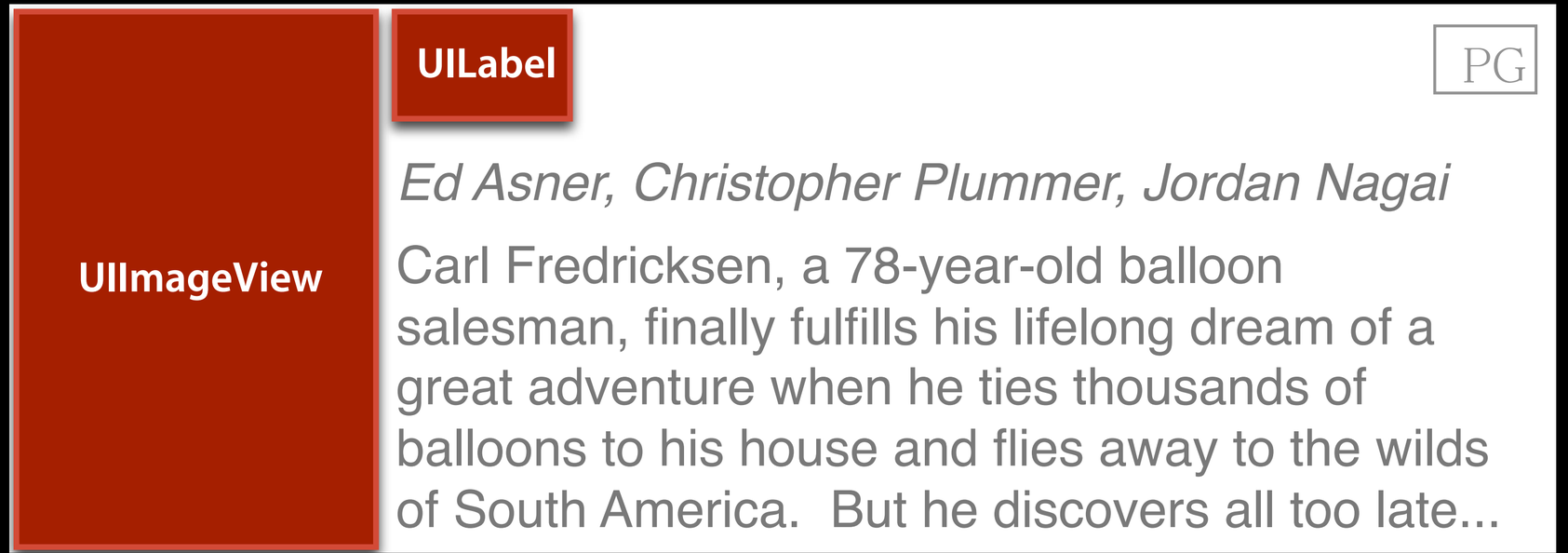
- Use opaque views when possible
- Avoid allocating views during scrolling
  - Allocating views is expensive



Drawing Optimizations

Improving your scrolling performance

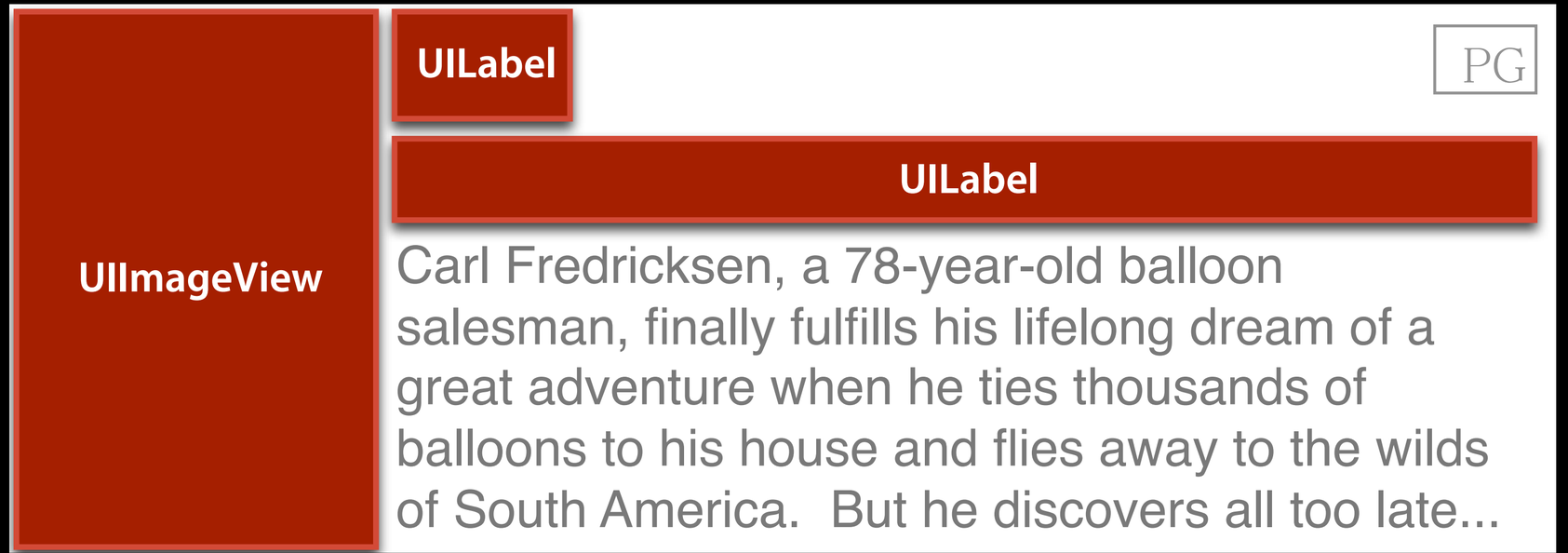
- Use opaque views when possible
- Avoid allocating views during scrolling
  - Allocating views is expensive



Drawing Optimizations

Improving your scrolling performance

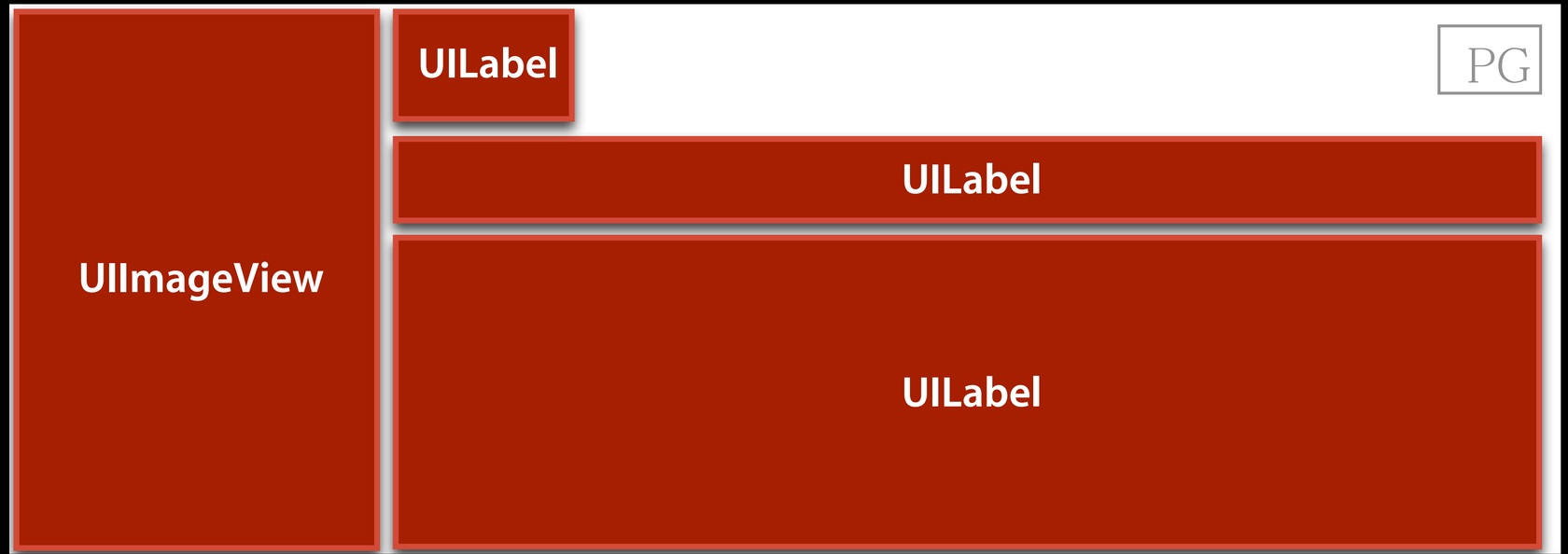
- Use opaque views when possible
- Avoid allocating views during scrolling
  - Allocating views is expensive



Drawing Optimizations

Improving your scrolling performance

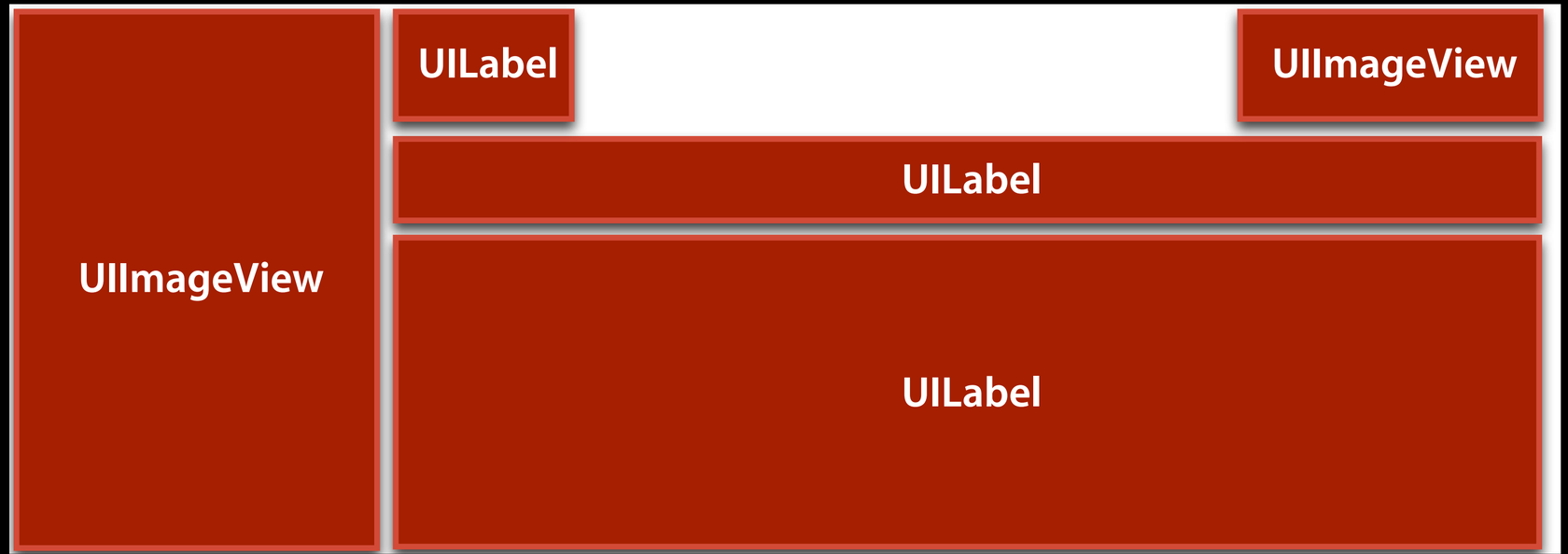
- Use opaque views when possible
- Avoid allocating views during scrolling
  - Allocating views is expensive



Drawing Optimizations

Improving your scrolling performance

- Use opaque views when possible
- Avoid allocating views during scrolling
  - Allocating views is expensive



Drawing Optimizations

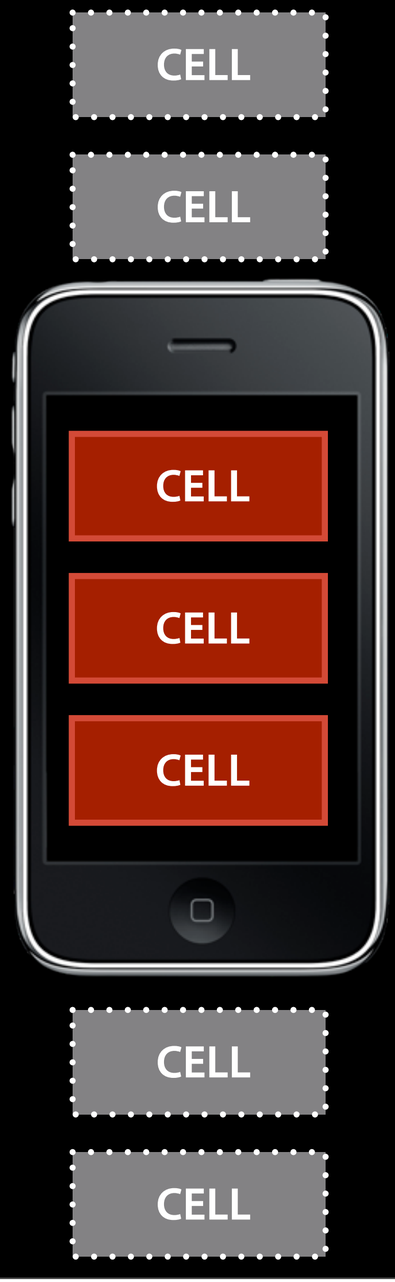
Improving your scrolling performance

- Use opaque views when possible
- Avoid allocating views during scrolling
  - Allocating views is expensive
- Reuse table cells
  - Tag cells with reuse IDs
  - Dequeue reusable cells in `tableView:cellForRowAtIndexPath:` and update with new content

Drawing Optimizations

Improving your scrolling performance

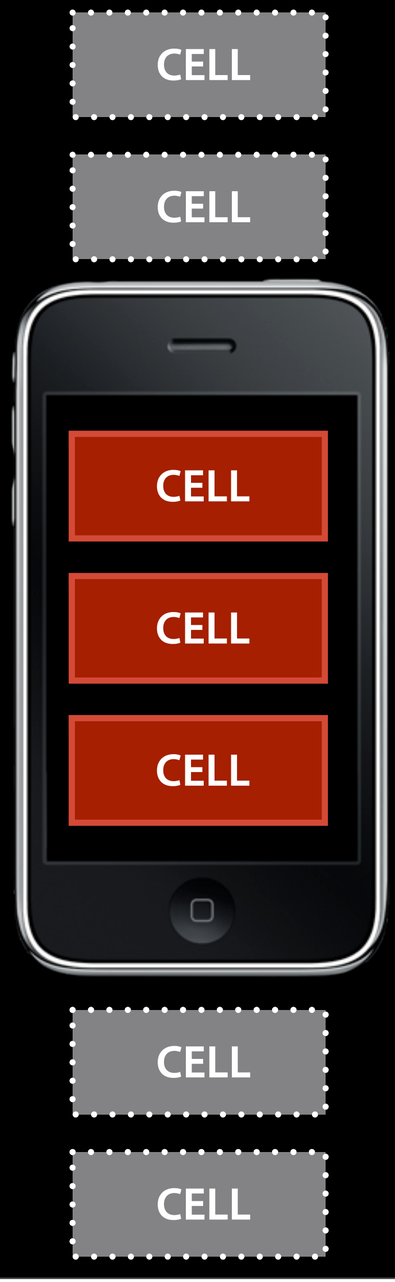
Before



Drawing Optimizations

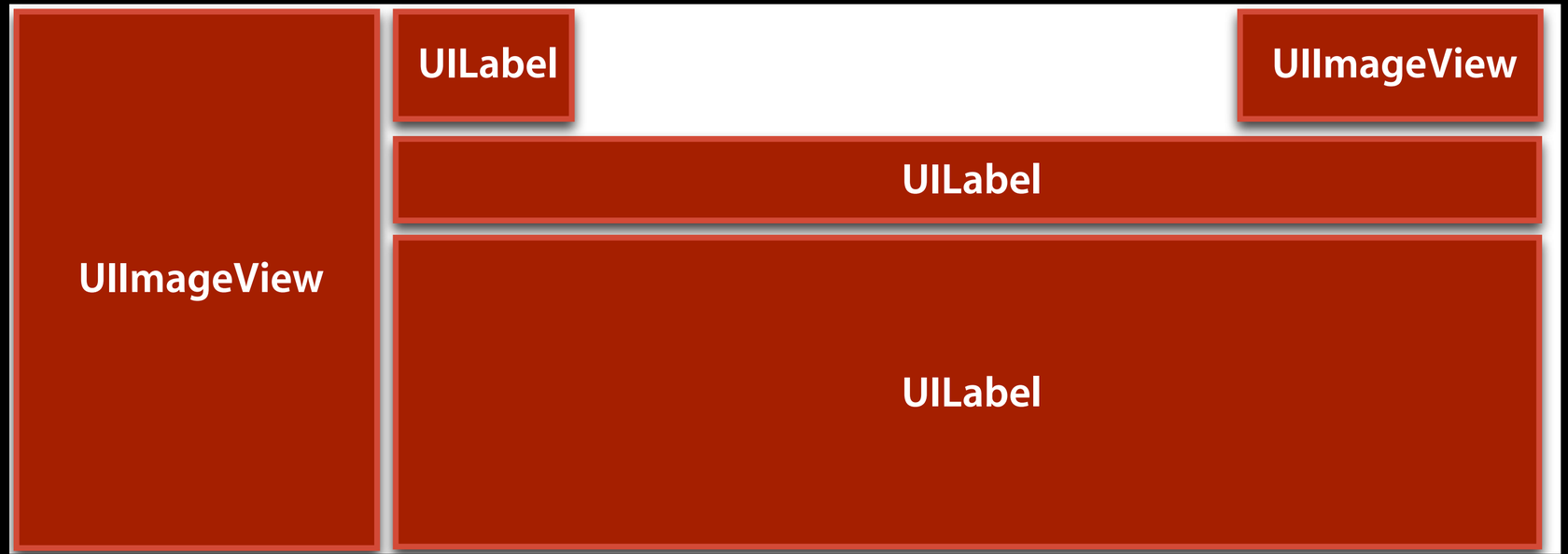
Improving your scrolling performance

After



Drawing Optimizations

Collapsing the cell view hierarchy

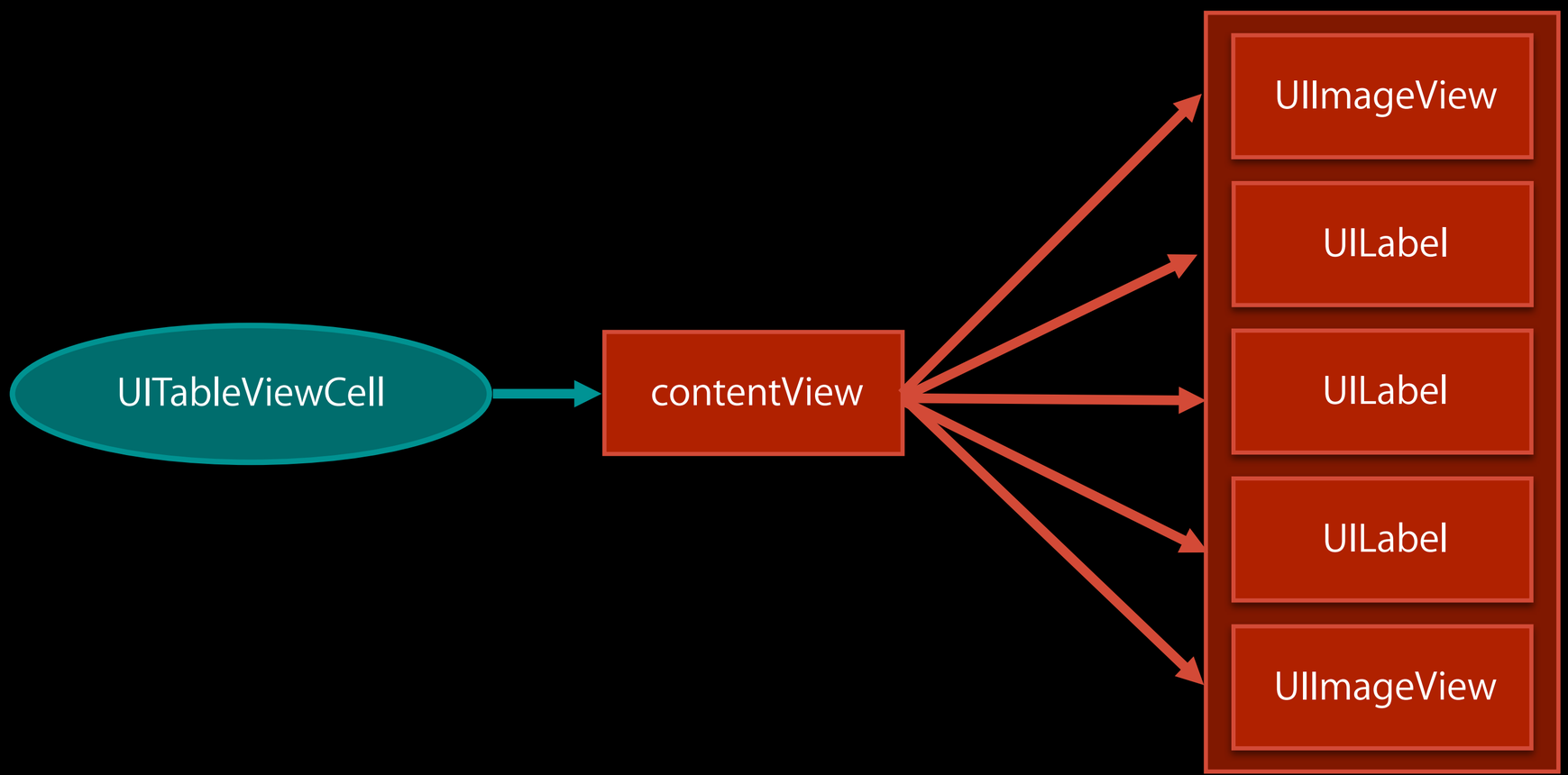


Drawing Optimizations

Collapsing the cell view hierarchy

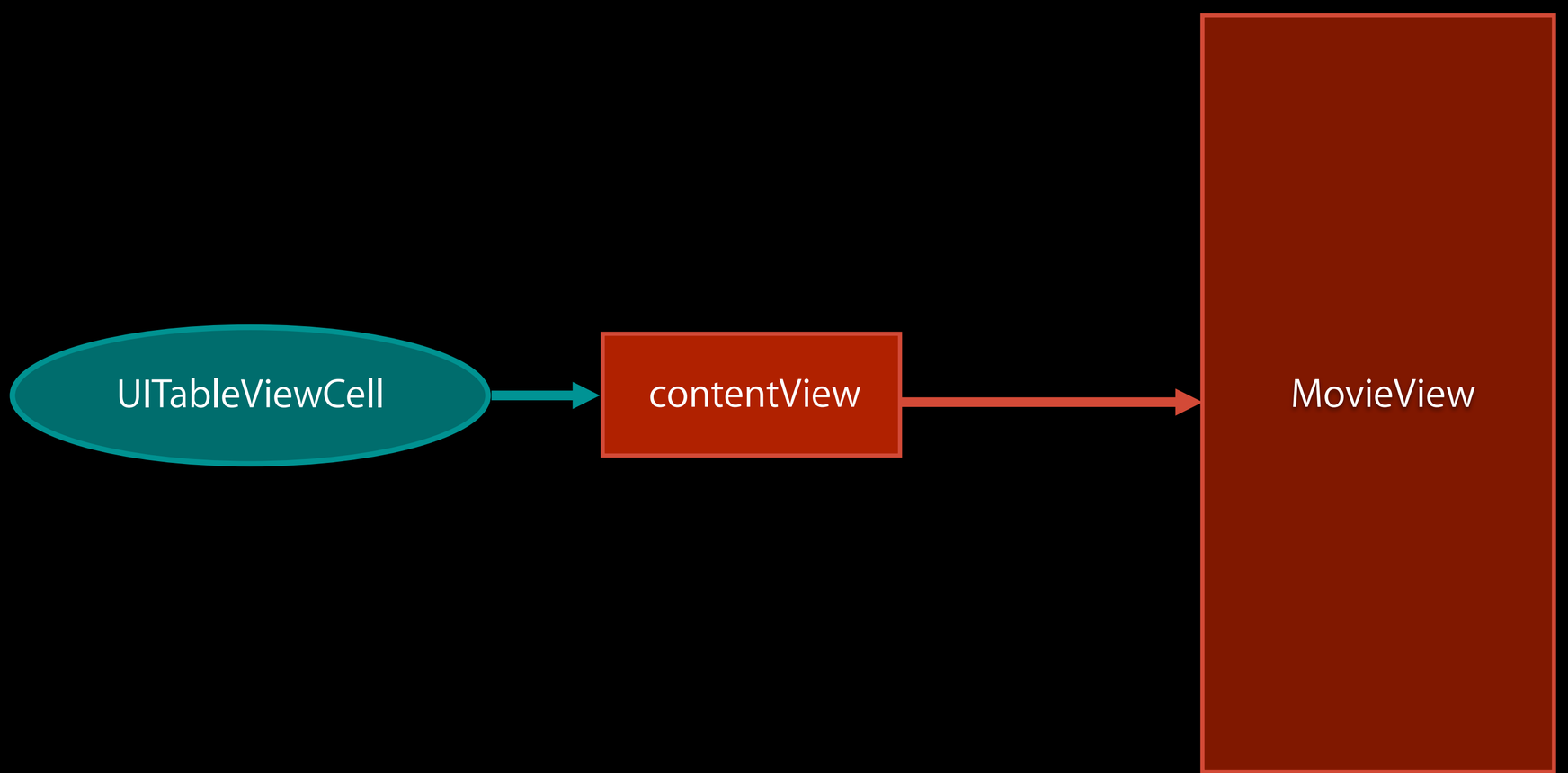
Drawing Optimizations

Collapsing the cell view hierarchy



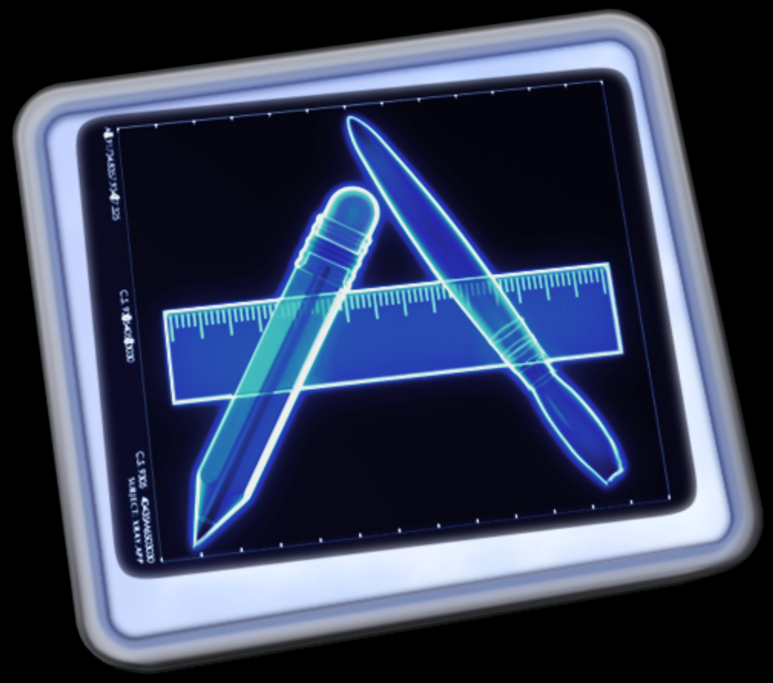
Drawing Optimizations

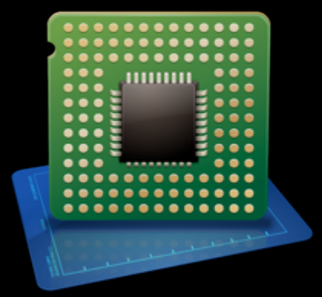
Collapsing the cell view hierarchy

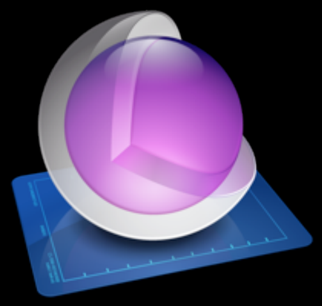


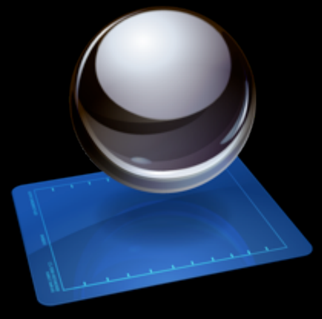
Tuning for Performance

Instruments









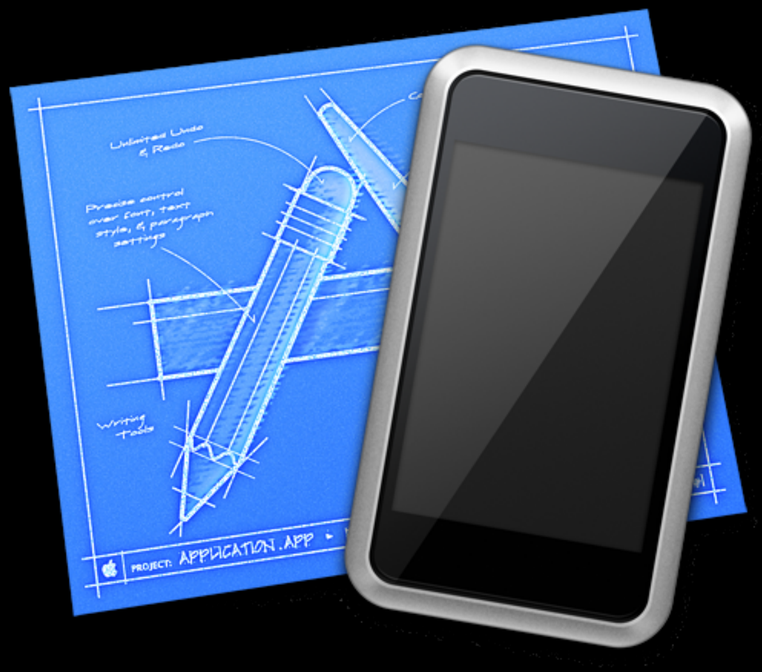
Tuning for Performance

Shark

- Time profile
- Windowed time facility

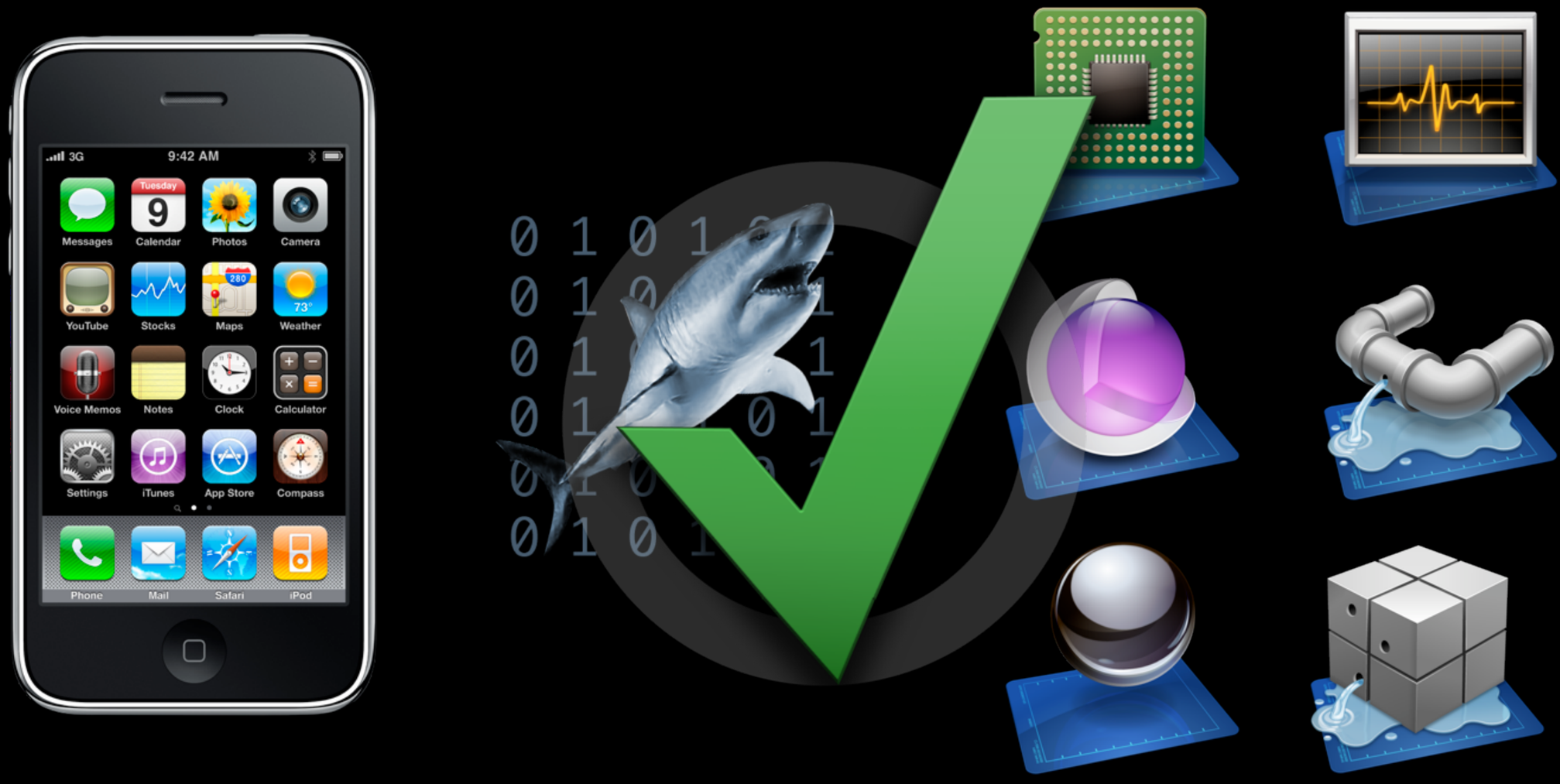
Tuning for Performance

Choose the right tuning environment



Tuning for Performance

Choose the right tuning environment



Demo

Improving scrolling performance

Drawing and Scrolling

Summary

- Mark views as opaque
- Draw minimally
- Reuse table cells
- Use PNGs
- Make optimal use of hardware

What You'll Learn

- Drawing and scrolling
- Application launch
- Memory usage
- Files and data
- Power and battery life

Application Launch

Design apps for fast launch, short use

- Consider the user's immediate needs
- Make your app launch and quit REALLY fast
  - Get your UI on screen as quickly as possible

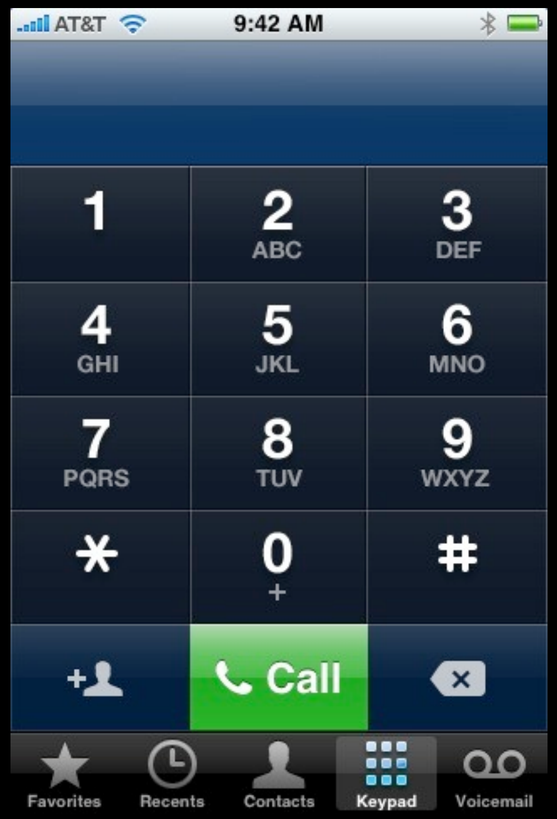
Application Launch

Design apps for fast launch, short use

- Load data lazily
- Use a small nib for initial UI
- Load only images that you need
- Don’t perform network operations in `-applicationDidFinishLaunching:`

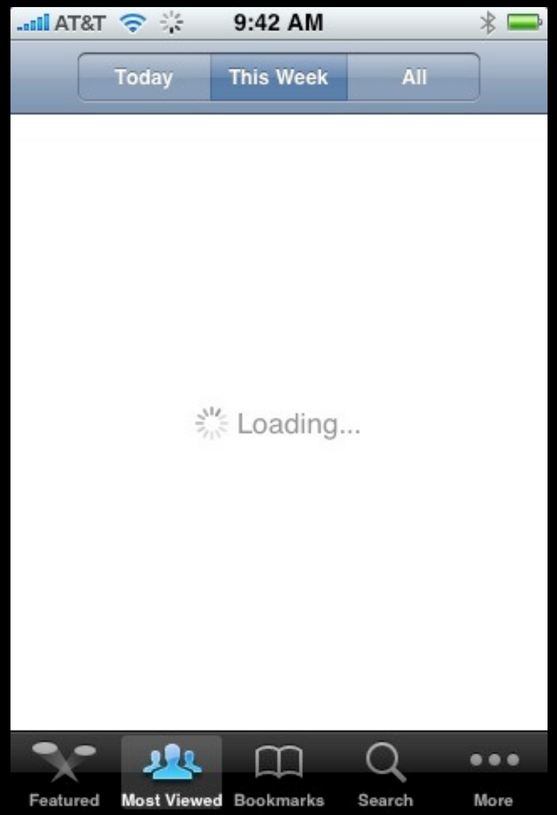
Phone

Loads only the visible UI



YouTube

Defers networking operations



What You'll Learn

- Drawing and scrolling
- Application launch
- Memory usage
- Files and data
- Power and battery life

About the Memory System

The virtual memory system

- It's there, but…
  - No swap file
  - Writable pages stay in memory
  - Read-only pages can be evicted and reloaded
- Supports mapping files into memory (`mmap`, `munmap`)

The Virtual Memory System

Managed memory model

- Objective-C
  - alloc/init to create objects
  - retain/release/autorelease to manage reference counts
- No garbage collection
- Standard C, C++ memory management

Memory Optimization Tips

The highlights

- Static memory
- Dynamic memory
- Application footprint
- Memory warnings

Static Memory

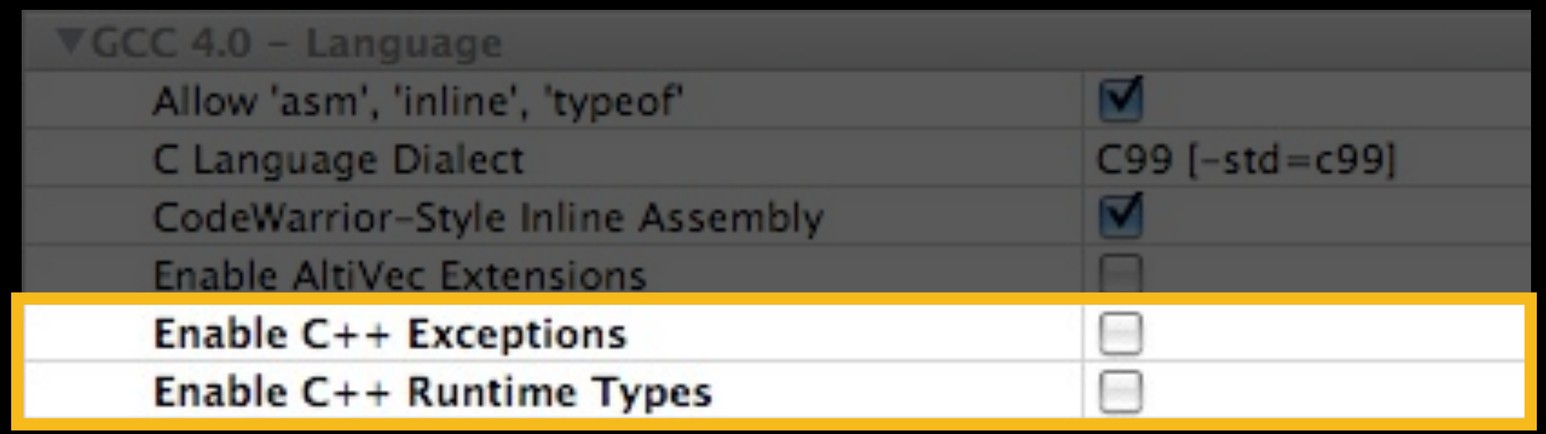
Reduce code size

	Thumb	Thumb-2
Instruction size	16-bit	16-, 32-bit
Floating point access	No	Yes
Use all the time?	No	Yes

Static Memory

Do not use compiler options you do not need

- Avoid C++ exceptions
- Avoid Runtime Type Identification (RTTI)



Dynamic Memory

Avoid atomic properties

- Declare properties as nonatomic

```
@property (nonatomic, retain) NSString* myString;
```

- Access is up to 10 times faster
- Not thread safe, however

Dynamic Memory

Allocate wisely

- Cache the right objects
  - Cache frequently used objects
  - But, be prepared to release those objects if needed
- Free objects when you are done with them
- Monitor object allocations with ObjectAlloc instrument

Dynamic Memory

Avoid autoreleasing objects

- Autorelease can be expensive
  - Use release whenever possible
- Use direct allocation instead of convenience methods

```
NSString* aString = [[NSString alloc] initWithFormat:@”Text”];
[myDictionary setObject:aString forKey:@”MyKey”];
[aString release];
```

Instead of:

```
NSString* aString = [NSString stringWithFormat:@”Text”];
[myDictionary setObject:aString forKey:@”MyKey”];
```

Dynamic Memory

If you must autorelease objects…

- Use local autorelease pools to prevent object accumulation
- Drain local autorelease pools periodically
  - Tight loops
  - XML parsing

```
NSAutoreleasePool *aPool = [[NSAutoreleasePool alloc] init];
for (i = 0; i < 1000; i ++) {
    // Create a bunch of autoreleased objects
}
[aPool drain];
```

Dynamic Memory

Eliminate memory leaks

Low-Memory Warnings

Don’t ignore the signs

- You must respond to low-memory warnings!
  - Failure to respond can cause app termination
- Other apps may cause low-memory conditions

Low-Memory Warnings

Receiving low-memory warnings

- Your app delegate object
  - `applicationDidReceiveMemoryWarning:` method
- `UIViewController` subclasses
  - Offscreen views may be purged automatically
  - Override `didReceiveMemoryWarning` method to release additional data
- Direct notifications
  - `UIApplicationDidReceiveMemoryWarningNotification`

Low-Memory Warnings

Taking action

- Purge off-screen views
- Delete cached objects
- Unload cached resource files

Memory

Summary

- Use the right compiler settings
- Minimize object lifetimes
- Eliminate memory leaks
- Respond to low-memory notifications

What You'll Learn

- Drawing and scrolling
- Application launch
- Memory usage
- Files and data
- Power and battery life

File Performance

Reading data

- Use memory mapping for large files
  - Avoids loading large files into memory
  - `mmap` and `munmap`
  - Cheap: `-[NSData initWithContentsOfMappedFile:]`
  - Expensive: `-[NSData initWithContentsOfFile:]`

File Performance

Core Data

- For large data sets, use Core Data
  - Doesn’t keep entire data set in memory
  - Loads “windows” of data in the background
  - Easy to create object models

File Performance

Download performance

- Don’t include unnecessary files
  - e.g., subversion directories

File Performance

Backup performance

- Data in ~/Documents is backed up
  - Use for user data but not for caches
- Use ~/Library/Caches for data that can be reacquired
  - Downloaded images

File Performance

Take advantage of built-in optimizations

- PNGs are optimized
- Plists are converted to compact binary format by Xcode
  - Use plists for structured data files

What You'll Learn

- Drawing and scrolling
- Application launch
- Memory usage
- Files and data
- Power and battery life

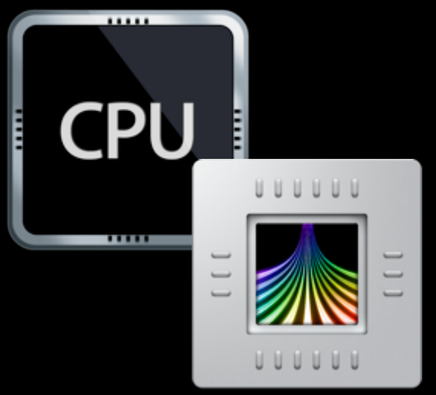
Power and Battery Life

Peter Handel
iPhone Power

Power Consumption

What we'll cover

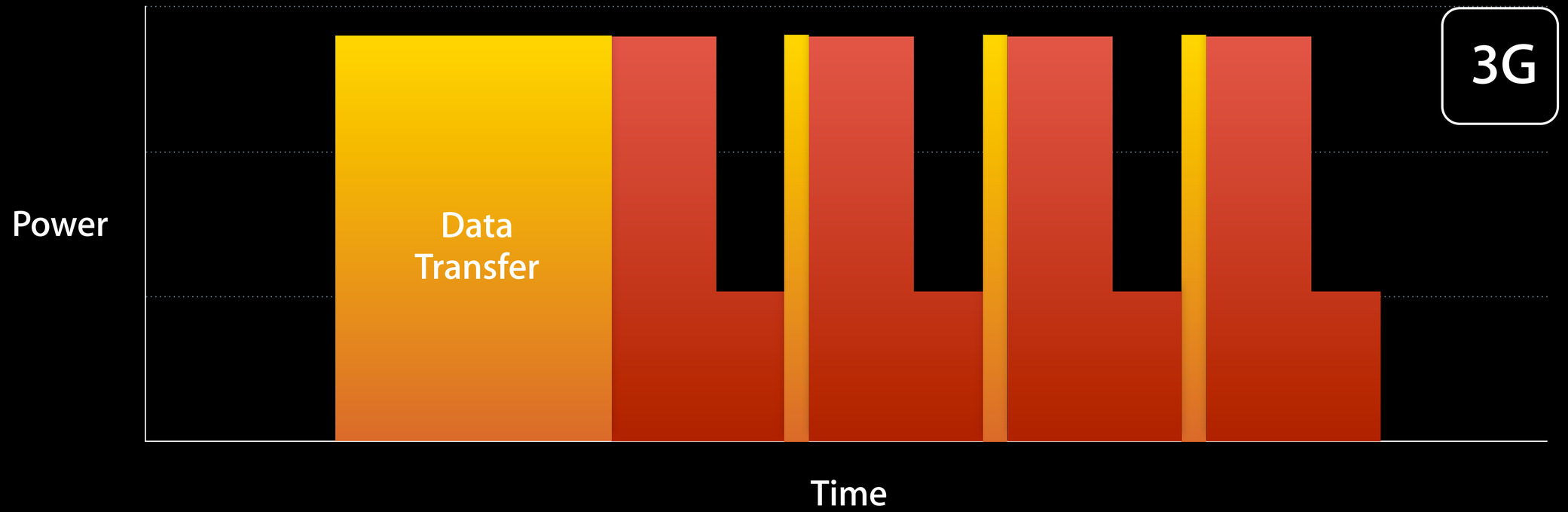
- Radio
- Core Location
- CPU/GPU



Power Consumption

Radios: 3G

- Very expensive to send data
- 3G networks require phones stay in high-power state for a few seconds after last packet is sent or received



Power Consumption

Radios: 3G

- Optimizing 3G transmissions
  - Coalesce data into large chunks, rather than thin stream
  - Minimize amount of data transmitted—Use compact data formats!
  - 3G radio chip: An idle chip is a green chip

Power Consumption

Radios: Wi-Fi

- Wi-Fi uses less power than 3G
  - …but it still uses a fair bit!
- Wi-Fi radios quiesce immediately after transmission

- Detect when you're on Wi-Fi versus cell
  - Example: More extreme data coalescence over cell

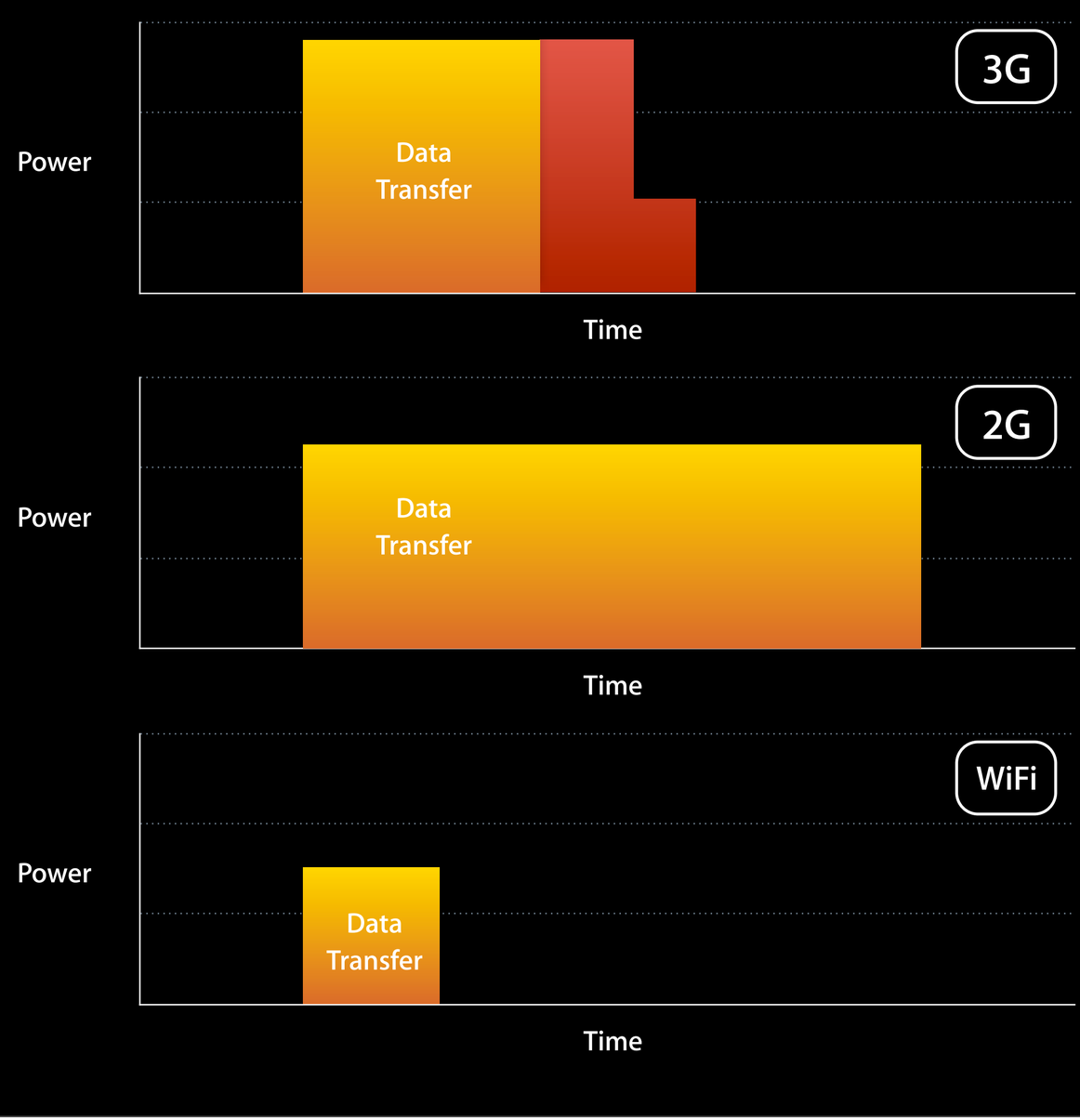
From SystemConfiguration:

```
if (flags & kSCNetworkReachabilityFlagsIsWWAN) {
```

Power Consumption

Radios: Wi-Fi vs. 3G vs. 2G

- 2G
  - Power consumption is between 3G and Wi-Fi
  - 2G network allows radios to idle immediately after data transfer



Power Consumption

What we'll cover

- Radio
- Core Location
- CPU/GPU

Power Consumption

Core Location

- Lots of apps use Core Location
- Lets you know where device is to varying degrees of accuracy

```
CLLocationManager *locationManager =                    alloc] init];
locationManager.desiredAccuracy = kCL                   dredMeters;
locationManager.distanceFilter = 100;
[locationManager startUpdatingLocatio
[locationManager stopUpdatingLocation];
```

Power Consumption

Core Location

- Use least amount of accuracy - default is `kCLLocationAccuracyBest`
  - GPS: `kCLLocationAccuracyNearestTenMeters`
  - Wi-Fi: `kCLLocationAccuracyHundredMeters`
  - Cell/Wi-Fi: `kCLLocationAccuracyKilometer`
  - Cell: `kCLLocationAccuracyThreeKilometers`

```
CLLocationManager *locationManager = [[CLLocationManager alloc] init];
locationManager.desiredAccuracy = kCLLocationAccuracyHundredMeters;
locationManager.distanceFilter = 100;
[locationManager startUpdatingLocation];
[locationManager stopUpdatingLocation];
```

Power Consumption

Core Location

- `distanceFilter` - dictates how often you receive location changed notifications
  - Set it appropriately
  - The default (`kCLDistanceFilterNone`) receives all movement updates
    - Can result in unnecessary events = higher CPU usage

```
CLLocationManager *locationManager = [[CLLocationManager alloc] init];
locationManager.desiredAccuracy = kCLLocationAccuracyHundredMeters;
locationManager.distanceFilter = 100;
[locationManager startUpdatingLocation];
[locationManager stopUpdatingLocation];
```

Power Consumption

Core Location

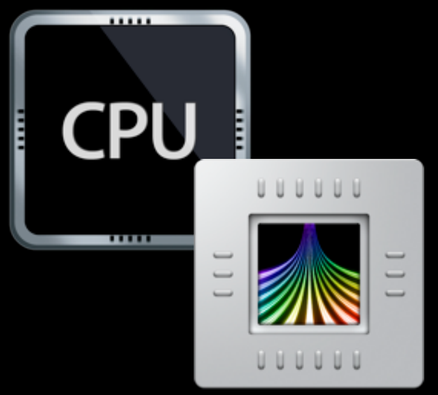
- Call `stopUpdatingLocation` after reaching desired accuracy
- CoreLocation manages GPS power for you
  - …so call `stopUpdatingLocation` as soon as you're finished
- GPS chip: An idle chip is a green chip

```
CLLocationManager *locationManager = [[CLLocationManager alloc] init];
locationManager.desiredAccuracy = kCLLocationAccuracyHundredMeters;
locationManager.distanceFilter = 100;
[locationManager startUpdatingLocation];
[locationManager stopUpdatingLocation];
```

Power Consumption

What we'll cover

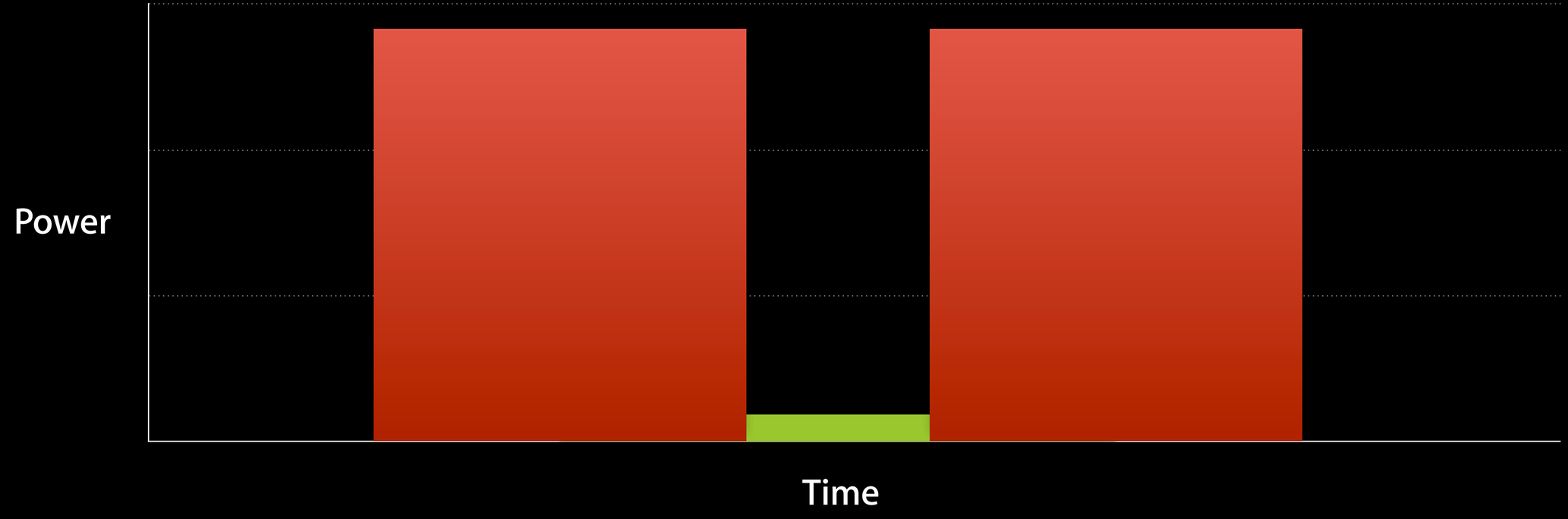
- Radio
- Core Location
- CPU/GPU



Power Consumption

CPU: Performance

- Improving performance results in better battery life
  - Fast code = less CPU time = less power
  - CPU: An idle chip is a green chip



Power Consumption

CPU: Polling vs. Events

- iPhone OS is event-based
  - You might want to poll a condition -> don't!
- Subscribe to events whenever possible
- If you must poll, use a timer with a low frequency

Power Consumption

CPU: Be bursty

- Be bursty! Consolidate CPU usage into short bursts
  - Allows CPU to enter idle state
  - May require code restructuring or different algorithm

Power Consumption

CPU: Procrastinate

- Delay work, possibly forever?
  - Example: When should a game write its state?

“The three chief virtues of a programmer are: Laziness, Impatience, and Hubris.” -Larry Wall

Power Consumption

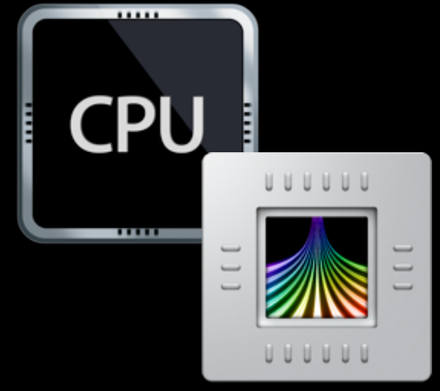
GPU

- When using OpenGL ES/GPU
  - Pick a fixed frame rate - 30 fps
  - If frame hasn't changed, don't redraw
    - Example: Chess game

Power Consumption

Summary

- Radios
  - Data transmission is expensive
  - Coalesce/compress data
- Core Location
  - Use least amount of accuracy you need
  - Unsubscribe from notifications when finished
- CPU/GPU
  - Optimizing for performance = optimizing for power
  - Be bursty, procrastinate
  - GPU: Use fixed frame rate (30fps), don't unnecessarily redraw frames
- An idle chip is a green chip



Summary

Advice so nice, we give it twice

- Drawing and scrolling
- Application launch
- Memory usage
- Files and data
- Power and battery life

Related Sessions

Understanding Memory Management on iPhone	Mission Thursday 3:30PM
iPhone Performance Optimization with Instruments	Marina Thursday 5:00PM
Performance Tuning with Shark on Mac and iPhone	North Beach Friday 9:00AM

Labs

iPhone Performance Lab	iPhone Lab A Tuesday 9:00AM
iPhone Memory Optimization Tutorial Lab	iPhone Lab A Thursday 12:00PM
Performance Tuning with Shark on Mac and iPhone Lab	iPhone Lab C Friday 2:00PM
iPhone Performance Lab	iPhone Lab B Friday 9:00AM
iPhone Performance Optimization with Instruments Lab	iPhone Lab B Friday 2:00PM

More Information

Michael Jurewitz
Developer Tools Evangelist
jurewitz@apple.com

Documentation
iPhone OS Programming Guide
http://developer.apple.com/iphone

Q&A

WWDC Session Survey

You can rate your sessions on the WWDC Attendee Site.

Survey forms are located on each session description page at:

developer.apple.com/wwdc/attendee

WWDC09